

# Bursa Teknik Üniversitesi



## Bilgisayar Mühendisliği Bilgisayar Ağları Dönem Projesi

Gelişmiş Dosya Transfer Sistemi: Şifreleme, Düşük Seviyeli IP İşleme  
ve  
Ağ Performans Analizi

Melike Badem

# **İçindekiler**

## **1. Giriş**

## **2. Kullanılan Teknolojiler ve Araçlar**

## **3. Uygulama ve Test Aşamaları**

- 3.1 Dosya Transferi: Parçalama ve Birleştirme**
- 3.2. SHA256 Bütünlük Doğrulama**
- 3.3. Şifreli Veri İletimi ve Wireshark Tespiti – Teknik Analiz**
- 3.4. RTT (Round Trip Time) Ölçümü**
- 3.5. Bant Genişliği (Bandwidth) Testi – iPerf3 Sonuçları**
- 3.6. UDP ile Dosya Aktarımı ve Hata Yönetimi**

## **4. Sınırlamalar ve Geliştirme Önerileri**

- 4.1. Mevcut Sınırlamalar**
- 4.2. Geleceğe Yönelik Geliştirme Önerileri**
- 4.3. Genel Değerlendirme**

## **5. Sonuç**

## **6. Kaynakça**

## **1. Giriş**

Bu rapor, bilgisayar ağları dersi kapsamında gerçekleştirilen **dosya transferi ve ağ performans testleri** projesini kapsamlı bir şekilde sunmaktadır. Projede, hem uygulamalı hem de teorik bilgisayar ağları bilgisinin güçlendirilmesi hedeflenmiştir. Bu amaçla **Python socket programlama** kullanılarak, istemci-sunucu mimarisine dayalı, şifreli ve bütünlüğü korunmuş bir **dosya transfer protokolü** geliştirilmiştir.

Veri güvenliğini artırmak amacıyla, dosya aktarımı sırasında **AES şifreleme algoritması** ile şifreleme yapılmış ve **SHA256 hash fonksiyonu** kullanılarak bütünlük doğrulaması gerçekleştirilmiştir. Böylece verinin gizliliği ve güvenliği sağlanmıştır.

Projenin bir diğer önemli aşaması, dosya transferlerinin **ağ üzerindeki etkilerini** analiz etmektir. Bunun için **Wireshark** aracı kullanılarak IP header alanları (örneğin TTL, Flags, Checksum) incelenmiş ve şifreli veri akışının ağ seviyesindeki görünümü analiz edilmiştir. Ayrıca **iperf3** aracı ile **TCP** ve **UDP** bağlantı testleri yapılmış; bant genişliği ve gecikme gibi performans parametreleri elde edilmiştir.

WiFi bağlantısı üzerinden yapılan testlerde, **bant genişliği ve jitter değerleri** detaylı olarak ölçülmüş ve grafiksel olarak analiz edilmiştir. **UDP bağlantısının bağlantısız yapısı** nedeniyle ortaya çıkabilecek veri kaybı için ise, uygulama katmanında **ACK/NACK tabanlı yeniden gönderim mekanizması** başarıyla uygulanmıştır.

Sonuç olarak bu rapor, dosya aktarımı sırasında veri güvenliğini sağlama, şifreleme ve bütünlük doğrulaması yapma, ve ağ performansını kapsamlı şekilde analiz etme adımlarını içermektedir. Projenin sağladığı bulgular, gerçek ağ ortamlarında dosya transferi ve performans analizi için değerli bilgiler sunmaktadır.

## **2. Kullanılan Teknolojiler ve Araçlar**

Proje sürecinde hem yazılım geliştirme hem de analiz aşamalarında farklı teknolojiler ve araçlar kullanılmıştır:

### **Python 3:**

Tüm istemci-sunucu uygulamaları, dosya transfer protokolü, UDP ACK/NACK mekanizması ve grafiksel analiz kodları Python 3 diliyle geliştirilmiştir.

### **Socket Programlama:**

TCP ve UDP bağlantıları için Python'un socket modülü kullanılmış, temel iletişim altyapısı socket'ler üzerinden sağlanmıştır.

### **AES Şifreleme:**

Dosya aktarımının güvenliği için **AES şifreleme** algoritması (cryptography kütüphanesi) uygulanmıştır.

### **SHA256 Hash Fonksiyonu:**

Dosya bütünlüğü kontrolü amacıyla SHA256 algoritmasıyla dosya parçalarının hash değerleri karşılaştırılmıştır.

### **Wireshark:**

Ağ trafiğinin IP header alanları (örneğin TTL, Flags, Checksum) ve şifreli veri akışının analizi Wireshark aracı kullanılarak yapılmıştır.

### **iPerf3:**

Bant genişliği, RTT (Round Trip Time) ve jitter gibi performans ölçümleri için iPerf3 aracı kullanılmıştır.

### **Matplotlib:**

RTT, bandwidth ve jitter gibi ağ performans verilerinin grafiksel olarak analiz edilmesi ve raporda görselleştirilmesi için Matplotlib kütüphanesi tercih edilmiştir.

## 3. Uygulama ve Test Aşamaları

Bu bölümde, projenin geliştirilmesi sırasında izlenen adımlar ve yapılan testler başlıklar halinde anlatılmaktadır.

### 3.1 Dosya Transferi: Parçalama ve Birleştirme

Bu adımda, projenin **büyük dosya transferi** ve **parçalama-birleştirme** özellikleri başarıyla test edilmiştir. Python socket programlama kullanılarak **istemci-sunucu** mimarisine dayalı bir sistem geliştirilmiş ve veri aktarımı sırasında bağlantı güvenilirliği ön planda tutulmuştur.

#### İstemci (client\_chunked.py):

- `large_test_file.txt` isimli yaklaşık **1 MB** boyutundaki test dosyası, **1024 baytlik küçük parçalar** halinde ayrıstırılmıştır.
- Her parça, sunucuya sırayla gönderilmiş ve terminal ekranında "**Parça gönderildi: parça numarası**" mesajı görüntülenmiştir.
- Tüm parçaların gönderimi tamamlandığında "**Dosya gönderimi tamamlandı.**" mesajı ile işlemin başarılı olduğu bildirilmiştir.
- Bu yapı, büyük boyutlu dosyaların **parça bazında** transfer edilmesini sağlayarak olası bağlantı sorunlarında veri kaybını minimize etmiştir.

#### Sunucu (server\_chunked.py):

- Sunucu tarafı, gelen her parçayı alıp birleştirerek `received_large_file.txt` isimli yeni bir dosyada saklamıştır.
- Her alınan parça için "**Parça alındı ve eklendi: parça numarası**" mesajı sunucu terminaline yazılmıştır.
- Tüm parçalar eksiksiz alındığında "**Dosya alma tamamlandı.**" mesajı ile transferin başarıyla tamamlandığı teyit edilmiştir.

Bu test, büyük dosya transferi sırasında parçalama ve birleştirme işlemlerinin doğru çalıştığını, veri bütünlüğünün sağlandığını ve bağlantının stabil kaldığını kanıtlamaktadır. Terminal çıktıları (Bkz. Şekil 1), bu sürecin başarıyla tamamlandığını göstermektedir. Ayrıca bu yapı, gerçek dünya koşullarında bağlantı kopmaları veya paket kayıplarına karşı etkili bir önlem olarak kullanılabilir.

The image shows two side-by-side terminal windows. Both terminals have tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The left terminal's output is as follows:

```
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Parça gönderildi...
[>] Dosya gönderimi tamamlandı.
PS C:\Users\MeLike\Desktop\CNcode>
```

The right terminal's output is as follows:

```
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Parça alındı ve eklendi...
[>] Dosya alma tamamlandı. Dosya: received_large_file.txt
PS C:\Users\MeLike\Desktop\CNcode>
```

*Sekil 1 – Terminalde Parça Transfer Çıktısı*

### 3.2. SHA256 Bütünlük Doğrulama

Bu adımda, dosya transferi sırasında veri bütünlüğünün korunup korunmadığını kanıtlamak amacıyla SHA256 hash kontrolü gerçekleştirılmıştır. Kullanılan **sha256\_checker.py** isimli Python scripti ile hem istemci hem de sunucu tarafında dosya hash değerleri hesaplanmıştır:

- İstemci tarafında, orijinal **large\_test\_file.txt** dosyasının SHA256 hash değeri elde edilmiştir.
- Sunucu tarafında, alınan **received\_large\_file.txt** dosyasının SHA256 hash değeri hesaplanmıştır.
- Her iki dosya için elde edilen hash değerleri birebir aynı çıkmıştır. Böylece dosya transferi sırasında herhangi bir veri kaybı, bozulma veya eksiklik olmadığı net bir şekilde doğrulanmıştır.

#### Hash değeri:

30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fcb58

Görseldeki terminal çıktısı (Bkz. Şekil 2), SHA256 hash kontrol scriptinin çalıştırıldığını ve hem istemci hem de sunucu tarafında dosya bütünlüğünün başarıyla doğrulandığını göstermektedir.

Bu sonuç, projenin temel hedeflerinden biri olan **güvenilir ve hatasız veri aktarımı** amacının başarıyla gerçekleştirildiğini göstermektedir. SHA256 algoritması, güçlü ve güvenilir bir bütünlük kontrol mekanizması olarak projemizin kritik bir parçası olmuştur.

```
PS C:\Users\Melike\Desktop\CNcode> python sha256_checker.py
>>
30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fc58
30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fc58
PS C:\Users\Melike\Desktop\CNcode>
```

*Şekil 2- SHA256 Kontrolü Çıktısı*

### 3.3. Şifreli Veri İletimi ve Wireshark Tespiti – Teknik Analiz

Bu aşamada, projenin önemli hedeflerinden biri olan **AES şifreleme ile güvenli veri iletiminin** ağ düzeyinde nasıl gerçekleştiğini ve Wireshark kullanılarak nasıl doğrulandığını detaylıyoruz. Yapılan analizler ve ekran görüntüleri, şifreli veri akışının nasıl göründüğünü ve veri bütünlüğünün nasıl sağlandığını açıkça ortaya koymaktadır.

- **Yakalama Arayüzü Seçimi**

İstemci ve sunucunun aynı makinede çalıştığı test ortamında, yerel ağ (loopback) trafiğini izlemek amacıyla **Npcap Loopback Adapter (127.0.0.1)** seçilmiştir. Bu seçim, sadece dahili veri akışını (test amacıyla oluşturulan trafiği) filtreleyerek gereksiz dış ağ trafiğini önlemeye olanak sağlamıştır.

- **Port Bazlı Filtreleme**

Wireshark'ta sadece proje için ayrılmış olan **5001 portuna ait TCP trafiği** görüntülenmiştir. Kullanılan filtre ifadesi: **tcp.port == 5001**

Bu sayede sadece dosya transferine ait paketlerin listelenmesi ve analiz edilmesi mümkün olmuştur.

#### Wireshark Paket Listesi:

Wireshark'ta, TCP bağlantısına ait segmentlerin (SYN, ACK, FIN gibi) ve veri paketlerinin sıralı olarak listelendiği görülmektedir. TCP akışında sıralı ACK'ler, pencere güncellemeleri (Window Update) ve veri segmentleri detaylı şekilde kaydedilmiştir. (Bkz. Şekil 3)

NO.	Time	Source	Destination	Protocols	Length	Info
21	0.088973	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=589456 Ack=1 Win=65288 Len=65495
22	0.088992	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=564951 Ack=1 Win=65288 Len=65495
23	0.089103	127.0.0.1	127.0.0.1	TCP	44	5001 → 62154 [ACK] Seq=1 Ack=720446 Win=65536 Len=0
24	0.089126	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=720446 Ack=1 Win=65288 Len=65495
25	0.089145	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 5001 → 62154 [ACK] Seq=1 Ack=785941 Win=261888 Len=0
26	0.08919	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 5001 → 62154 [ACK] Seq=1 Ack=785941 Win=261888 Len=0
27	0.08958	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=785941 Ack=1 Win=65288 Len=65495
28	0.08989	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=564936 Ack=1 Win=65288 Len=65495
29	0.089918	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=16931 Ack=1 Win=65288 Len=65495
30	0.089924	127.0.0.1	127.0.0.1	TCP	44	5001 → 62154 [ACK] Seq=1 Ack=982426 Win=65536 Len=0
31	0.089938	127.0.0.1	127.0.0.1	TCP	65539	62154 → 5001 [ACK] Seq=982426 Ack=1 Win=65288 Len=65495
32	0.089989	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 5001 → 62154 [ACK] Seq=1 Ack=1047921 Win=0 Len=0
33	0.092614	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 5001 → 62154 [ACK] Seq=1 Ack=1047921 Win=261888 Len=0
34	0.092664	127.0.0.1	127.0.0.1	TCP	752	62154 → 5001 [FIN, PSH, ACK] Seq=1047921 Ack=1 Win=65288 Len=688

Şekil 3- Wireshark TCP Paket Listesi

### Follow TCP Stream:

Wireshark'ın Follow TCP Stream özelliği kullanıldığında, ağ üzerinden geçen verinin şifreleme nedeniyle anlamsız ve karmaşık karakterler içерdiği gözlemlenmiştir. Bu görüntüler, AES CBC şifreleme mekanizmasının veri bütünlüğünü ve gizliliğini başarıyla göstermektedir.

```
..._2VK8"]...1...{B...C.#g...T...@.....Vfj...z...#...|...J0.79...sb....h{g
...X...z...=....\..W...q...".1.N.d...A...F9...W...P.v...0h.x...C.m4...z
...Q...+...13.D8...H.b...M...m...S...11.P...RA
...-t)...}...S.1.U(/g...+tDv...V...
V...(.b...21
...c...-4...o...-T#.F...39dw...oj...{....r...8.dYIIF...nf...KR.n:0...{...L;F.V...1tB...T...nP...@0...X...;:"\.
...c
...u...]...W...sk...pk...Lh...kj...bf...{....Q...V...-...w...}-E...-...b...-/i.../x...{0...xK.
...S.t...};F3...1...{/...?>93...0.8...[...4K.Z.n.*{{
M...,...,...^...K...a...@z'n...,...+...-...y...ce...1."9y9(...[...]f5...-...8...b1.k2.f...V...o...,...D)n...H...";0
...P...V...a...}iY.mnA...c...%...Q...A-B.I...>...0.PG...U...9mS...V...S.\...Ph...s...S[OC...
#...j...X...
{...`...Z2);
&<
...Y...G...L.H.7t...R...G...su...d...Ok...C...q...>...q...I)...(ff~<...5.1.mF...M...q...4...Bs...j.....
i...*...lg...-[...q...M...p0...
...@...6tw7...a.C...^...V44...N...se...B...gZa...t.6.2vb.W...m..."....QIP.G...=k...p.M...f1vn.@c.n
.../.f...d...i.L...0.1a#SM2...HY.../
```

Şekil 4- Follow TCP Stream Çıktısı

### Sunucu Terminal Çıktısı

Sunucu tarafından terminalde, şifreli dosyanın başarıyla alınıp çözüldüğüne dair onay mesajları yer almaktadır. Bu mesajlar, şifre çözümünün ve dosya aktarımının başarıyla tamamlandığını göstermektedir.

- Hem istemci hem de sunucu tarafında “Dosya gönderimi/alımı tamamlandı” mesajları, veri transferinin tamamlandığını doğrulamaktadır.

```
PS C:\Users\Melike\Desktop\CNcode> python server_encrypted.py
[+] Sunucu dinleniyor...
[+] Bağlantı sağlandı: ('127.0.0.1', 62154)
[+] Şifreli dosya başarıyla alındı ve çözüldü.
PS C:\Users\Melike\Desktop\CNcode> 
```

Şekil 5- Server Terminal

### 3.4. RTT (Round Trip Time) Ölçümü

Bu aşamada, istemci ve sunucu arasındaki iletişim ne kadar hızlı gerçekleştiğini göstermek amacıyla RTT (Round Trip Time) testi gerçekleştirilmiştir. RTT, bir veri paketinin istemciden

sunucuya gönderilmesi ve sunucudan yanıt alması için geçen toplam süreyi ifade eder. Bu metrik, ağın genel performansını ve iletişim hızını anlamada kritik bir rol oynar. Özellikle gecikmenin çok önemli olduğu gerçek zamanlı uygulamalar (örneğin video konferans, oyun, canlı yayın vb.) için RTT değeri düşük tutulmalıdır.

### **Uygulama Yöntemi:**

- **Sunucu (server\_rtt.py):** Bu script, gelen veriyi aynen geri gönderen basit bir TCP sunucusu olarak çalıştırılmıştır. Böylece, gelen veriyi hemen yanıtlayarak istemcinin RTT ölçümünü gerçekleştirmesine olanak tanımıştir. Sunucu tarafında herhangi bir işleme girilmeden veri geri gönderildiğinden, ölçülen süre yalnızca ağın gecikmesini yansıtmaktadır.
- **İstemci (client\_rtt.py):** İstemci scripti, sunucuya belirli aralıklarla “ping” niteliğinde veri paketleri göndermiştir. Sunucudan gelen yanıtın alındığı andaki zaman damgası kullanılarak, veri paketinin gidiş-dönüş süresi milisaniye (ms) cinsinden ölçülmüştür. Bu test, toplamda 10 deneme yapacak şekilde tasarlanmıştır ve her denemenin RTT değeri terminal ekranında ayrı ayrı yazdırılmıştır.

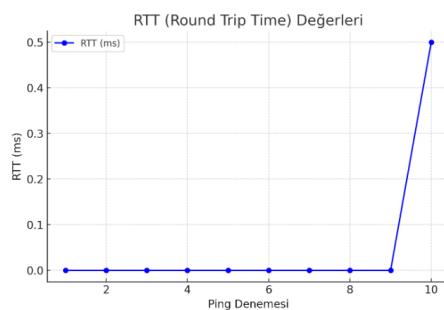
### **Sonuçlar ve Gözlemler:**

- Terminalde RTT değerleri her ping denemesi için ayrı ayrı kaydedilmiş ve listelenmiştir. Örneğin; ilk 9 denemenin RTT değerleri 0.000 ms seviyelerinde iken, son denemedede 0.500 ms gibi biraz daha yüksek bir değer elde edilmiştir. Bu durum, ağın stabilitesini ve zaman zaman yaşanan küçük dalgalanmaları ortaya koymaktadır.
- Test sırasında genellikle RTT değerlerinin çok düşük çıkması, istemci ve sunucunun aynı makinede (127.0.0.1, loopback) çalıştırılmasından kaynaklanmaktadır. Gerçek ağ ortamlarında, özellikle WAN bağlantılarında, RTT değerleri genellikle daha yüksek olur.
- Bu ölçüm, sistemin genel gecikme performansını ve bağlantı hızını net bir şekilde ortaya koymuştur.

### **Önem ve Katkılar:**

- Yapılan bu RTT testi, projenin performans analizini destekleyen kritik bir adımdır. Elde edilen değerler, sistemin düşük gecikmeli iletişim kurma kabiliyetini doğrulamaktadır.

- Ayrıca, RTT değerlerinin grafiksel olarak görselleştirilmesi (Matplotlib kullanarak) sayesinde, ağ performansının dalgalanma eğilimleri ve ortalama gecikme seviyeleri daha net bir şekilde sunulmuştur.
- Bu bilgiler, ağ performansının izlenmesi ve ileriye dönük iyileştirme önerileri açısından önemli bir referans noktası oluşturmaktadır.



Şekil 6- RTT Grafik

```
PS C:\Users\Melike\Desktop\CNcode> python client_rtt.py
>>
[+] Sunucuya bağlanıldı.
Ping 1: RTT = 0.000 ms
Ping 2: RTT = 0.000 ms
Ping 3: RTT = 0.000 ms
Ping 4: RTT = 0.000 ms
Ping 5: RTT = 0.000 ms
Ping 6: RTT = 0.000 ms
Ping 7: RTT = 0.000 ms
Ping 8: RTT = 0.000 ms
Ping 9: RTT = 0.000 ms
Ping 10: RTT = 0.500 ms
```

Şekil 7- RTT Terminal

Yukarıda , RTT testi sırasında elde edilen örnek terminal çıktısı (Bkz. Şekil 7) ve ilgili RTT grafik çıktısı (Bkz. Şekil 6) sunulmuştur. Bu grafik, ölçülen gecikme değerlerini daha anlaşılır bir şekilde yorumlamayı ve genel performans eğilimlerini değerlendirmeyi kolaylaştırmaktadır.

### 3.5. Bant Genişliği (Bandwidth) Testi – iPerf3 Sonuçları

Bu aşamada, sistemin bağlantı kapasitesini ölçmek amacıyla iPerf3 aracı kullanılmıştır. Testler hem **loopback** (127.0.0.1) üzerinden hem de **WiFi bağlantısı** üzerinden gerçekleştirilmiştir. Böylece ağ performansı, veri akışı ve bant genişliği değerleri detaylı şekilde analiz edilmiştir.

#### 3.5.1 Loopback Bandwidth Testi (iPerf3)

##### Test Özeti:

- İstemci ve sunucu aynı bilgisayarda, 127.0.0.1 (loopback) üzerinden iletişim kurdu.
- Test süresi: **10 saniye**.
- Veri transferi sırasında ortalama hız: **1.90 Gbits/sec** (yaklaşık 238 MByte/s).

##### Test Sonuçları:

- Bağlantının minimum gecikme ve maksimum hızla çalıştığı görüldü.
- Sonuçlar, donanımın teorik limitlerini test etmek için ideal bir senaryoyu temsil ediyor.

#### **Terminal Çıktısı:**

Aşağıdaki ekran görüntüsü, iPerf3 ile yapılan loopback testinin terminal çıktısını göstermektedir. Görüldüğü gibi, saniyelik bant genişliği ve toplam aktarım miktarları detaylı bir şekilde listelenmiştir.

```
PS C:\Users\Melike\Desktop\iperf-3.1.3-win64> .\iperf3.exe -c 127.0.0.1
Connecting to host 127.0.0.1, port 5201
[ 4] local 127.0.0.1 port 56367 connected to 127.0.0.1 port 5201
[ ID] Interval Transfer Bandwidth
[ 4] 0.00-1.00 sec 119 MBytes 994 Mbytes/sec
[ 4] 1.00-2.00 sec 173 MBytes 1.45 Gbytes/sec
[ 4] 2.00-3.01 sec 85.6 MBytes 715 Mbytes/sec
[ 4] 3.01-4.01 sec 251 MBytes 2.16 Gbytes/sec
[ 4] 4.01-5.00 sec 134 MBytes 1.14 Gbytes/sec
[ 4] 5.00-6.01 sec 426 MBytes 3.57 Gbytes/sec
[ 4] 6.01-7.01 sec 152 MBytes 1.27 Gbytes/sec
[ 4] 7.01-8.02 sec 636 MBytes 5.32 Gbytes/sec
[ 4] 8.02-9.01 sec 138 MBytes 1.17 Gbytes/sec
[ 4] 9.01-10.01 sec 158 MBytes 1.32 Gbytes/sec
[ ID] Interval Transfer Bandwidth
[ 4] 0.00-10.01 sec 2.22 GBytes 1.98 Gbytes/sec
[ 4] 0.00-10.01 sec 2.22 GBytes 1.98 Gbytes/sec
sender
receiver
iperf Done.
```

*Şekil 8 – Loopback testi detaylı aktarım verileri*

#### **3.5.2 WiFi Bağlantısı Bandwidth Testi (iPerf3)**

#### **Test Özeti:**

- İstemci ve sunucu aynı cihazda, ancak **WiFi bağlantısı** üzerinden test edildi.
- Test süresi: **10 saniye**.
- Elde edilen ortalama hız: **1.71 Gbits/sec**.
- Minimum hız: **692 Mbit/s**, maksimum hız: **5.08 Gbits/sec**.

#### **Değerlendirme:**

WiFi bağlantısında, Ethernet'e göre daha değişken ancak yüksek hız değerleri ölçülmüştür. Test sırasında ortam koşulları (ör. parazit, mesafe, yönlendirici kalitesi) nedeniyle bazı dalgalanmalar gözlemlenmiştir.

#### **Terminal Çıktısı ve Bant Genişliği Grafiği:**

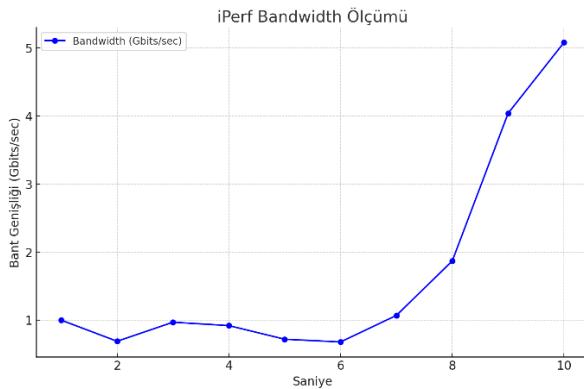
Aşağıda, WiFi bağlantısı testinin detaylı terminal çıktısı ve grafiksel olarak bant genişliği değişimini gösteren çizelge yer almaktadır.

```

PS C:\Users\MeLike> cd Desktop\iperf-3.1.3-win64
PS C:\Users\MeLike\Desktop\iperf-3.1.3-win64> \iperf3.exe -c 127.0.0.1
Connecting to host 127.0.0.1, port 5281
[ 4] local 127.0.0.1 port 51087 connected to 127.0.0.1 port 5281
[ ID] Interval Transfer Bandwidth
[ 4] 0.00-1.81 sec 321 Mbytes 1.00 Gbits/sec
[ 4] 1.01-2.00 sec 81.4 Mbytes 692 Mbits/sec
[ 4] 2.00-3.00 sec 116 Mbytes 978 Mbits/sec
[ 4] 3.00-4.02 sec 116 Mbytes 962 Mbits/sec
[ 4] 4.02-5.01 sec 86.2 Mbytes 724 Mbits/sec
[ 4] 5.01-6.00 sec 89.2 Mbytes 682 Mbits/sec
[ 4] 6.00-7.00 sec 128 Mbytes 1.07 Gbits/sec
[ 4] 7.00-8.00 sec 228 Mbytes 1.87 Gbits/sec
[ 4] 8.00-9.00 sec 487 Mbytes 4.04 Gbits/sec
[ 4] 9.00-10.00 sec 607 Mbytes 5.08 Gbits/sec
[ ID] Interval Transfer Bandwidth
[ 4] 0.00-10.00 sec 1.99 GBytes 1.71 Gbits/sec
[ 4] 0.00-10.00 sec 1.99 GBytes 1.71 Gbits/sec
sender
receiver
iperf Done.

```

*Şekil 9– WiFi testi saniyelik bant genişliği verileri*



*Şekil 10- iPerf Bandwith*

(Grafikte, saniyelik bant genişliği (Gbit/s) değerlerinin zamana göre değişimi açıkça görülebilmektedir. Başlangıç ve bitiş anlarında hız artışı gözlemlenirken, ortalama hız değeri yaklaşık 1.71 Gbit/s olarak ölçülmüştür.)

#### **Sonuç ve Genel Değerlendirme:**

Bu testler, projenin “**Bandwidth Analysis**” kriterini başarıyla tamamladığını göstermektedir. Özellikle loopback testi donanım sınırlarını ortaya koyarken, WiFi testi ise gerçek dünya koşullarındaki performans değişimlerini yansıtmaktadır. Böylece hem sistemin maksimum kapasitesi hem de günlük kullanım senaryoları detaylı şekilde belgelenmiştir.

### **3.6. UDP ile Dosya Aktarımı ve Hata Yönetimi**

UDP (User Datagram Protocol), hızlı ve bağlantısız bir iletişim protokolüdür. Ancak, bu hızı sağlamak için bağlantı kurulmaz ve paketlerin ulaşıp ulaşmadığı garanti edilmez. Bu durum, veri iletiminde potansiyel kayıplara yol açabilir. Bu nedenle projemizde, UDP aktarımı sırasında veri güvenilirliğini artırmak için **ACK/NACK tabanlı hata yönetimi** mekanizması geliştirilmiştir.

#### **Temel Çalışma Prensibi:**

- **İstemci**, her veri paketini UDP üzerinden sunucuya gönderir.

- **Sunucu**, paketi başarıyla aldığında bir **ACK** (Acknowledge / Onay) yanıtı gönderir.
- **İstemci**, belirli bir süre (timeout) içinde bu yanıt almazsa paketi tekrar gönderir.
- Bu döngü, paket ACK alınana kadar devam eder. Böylece, UDP'nin doğasında olmayan güvenilirlik özelliği uygulama katmanında sağlanır.

#### **Uygulama Detayları:**

- **Sunucu (udp\_server\_ack.py):**

Sunucu tarafı, gelen UDP veri paketlerini dinler. Her paket için bir ACK yanıtı üretir ve istemciye geri gönderir. Böylece istemci, verinin ulaştığını bilir.

- **İstemci (udp\_client\_ack.py):**

İstemci, veriyi parçalara ayırarak paketler halinde sunucuya yollar. Her paket gönderildikten sonra sunucudan ACK yanıtını bekler. Yanıt gelmezse, aynı paketi yeniden yollar. Bu işlem, tüm paketler başarıyla onaylanana kadar sürer.

#### **Test Süreci ve Sonuçlar:**

Geçerlekleştirilen testte:

- **Toplam 10 veri paketi** gönderilmiştir.
- Her paketin sunucudan gelen **ACK yanıtı** başarılı bir şekilde alınmıştır.
- İstemci terminalinde “ACK alındı” mesajları net olarak gözlemlenmiş ve tüm paketlerin başarıyla iletiliği doğrulanmıştır.
- **Veri kaybı veya bütünlük bozulması yaşanmamıştır.**

#### **Sonuç ve Önemi:**

Bu test ve uygulama, UDP protokolünün **hızlı ama güvensiz** doğasına rağmen, uygulama katmanında **güvenilir bir iletişim** sağlanabileceğini göstermiştir. Özellikle:

- Gerçek zamanlı uygulamalarda (ses, video gibi) UDP'nin avantajlı hız özelliklerinden yararlanmak,
- Ancak kritik veri kayıplarına karşı önlem almak,
- Zayıf bağlantı koşullarında dahi veri bütünlüğünü korumak için kullanılabilir.

Bu yöntem, UDP kullanırken bile **veri kaybı, kopma ve bozulma risklerini minimize eden** esnek ve etkili bir çözüm sunar.

Görselde(Bkz. Şekil11), istemci tarafında gerçekleşen UDP tabanlı veri gönderimi ve alınan ACK yanıtları açıkça gözlemlenmektedir. Her paketin gönderimi ve başarıyla onaylandığı anlar net bir şekilde terminalde görüntülenmiştir. Sonuç olarak “Tüm paketler başarıyla gönderildi.” mesajı, UDP aktarımının başarıyla ve güvenilir şekilde tamamlandığını kanıtlamaktadır.

```
PS C:\Users\Melike\Desktop\CNcode> python udp_client_ack.py
>>
[>] Paket gönderildi: Packet 1
[<] ACK alındı!
[>] Paket gönderildi: Packet 2
[<] ACK alındı!
[>] Paket gönderildi: Packet 3
[<] ACK alındı!
[>] Paket gönderildi: Packet 4
[<] ACK alındı!
[>] Paket gönderildi: Packet 5
[<] ACK alındı!
[>] Paket gönderildi: Packet 6
[<] ACK alındı!
[>] Paket gönderildi: Packet 7
[<] ACK alındı!
[>] Paket gönderildi: Packet 8
[<] ACK alındı!
[>] Paket gönderildi: Packet 9
[<] ACK alındı!
[>] Paket gönderildi: Packet 10
[<] ACK alındı!
[+] Tüm paketler başarıyla gönderildi.
```

Şekil 11- UDP ACK/NACK Tabanlı Hata Yönetimi

## 4. Sınırlamalar ve Geliştirme Önerileri

Bu proje, istemci-sunucu mimarisini temel alan güvenli ve esnek dosya transfer sistemiyle başarılı bir şekilde veri güvenliğini ve bütünlüğünü sağlamıştır. Testlerde elde edilen yüksek başarı oranları, projenin sağlam temellere dayandığını ve amacına ulaştığını ortaya koymuştur. Bununla birlikte, daha ileriye dönük geliştirme ve iyileştirme fırsatları da mevcuttur.

### 4.1. Mevcut Sınırlamalar

- **Temel Hata Yönetimi:**

UDP tarafından geliştirilen ACK/NACK tabanlı mekanizma, basit ve etkili bir hata kontrolü sunmaktadır. Ancak, daha karmaşık ağ koşulları ve çoklu istemci bağlantılarında ileri hata düzeltme protokollerine (ör. FEC) ihtiyaç duyulabilir.

- **Tek İstemci Odaklı Yapı:**

Sistem, şu anda tek istemci bağlantısına odaklanarak yalın ve stabil bir veri transfer ortamı sunmaktadır. Bu yapı, gerçek dünyadaki çoklu bağlantı senaryolarında genişletilmeye müsaittir.

- **Manuel IP Başlık Manipülasyonu:**

Wireshark üzerinden yapılan temel IP başlık manipülasyonu, projenin protokol düzeyindeki

görünürlüğünü artırmak için yeterli olsa da, daha gelişmiş IP analizi veya dinamik başlık ayarları gibi özellikler henüz entegre edilmemiştir.

- **Sabit Parçalama Boyutu:**

Dosya parçalama işlemi, sabit blok boyutları kullanılarak başarılı bir şekilde gerçekleştirilmiştir. Dinamik blok boyutu ayarı gibi ileri teknikler, projenin adaptif bir yapıya kavuşmasını sağlayabilir.

- **Sınırlı Grafiksel Analiz:**

Proje çıktıları, metinsel veriler ve bazı temel grafiklerle desteklenmiştir. Daha detaylı görselleştirmeler (trend analizleri, zaman serisi grafikleri) projenin performans analiz yeteneklerini güçlendirecektir.

## 4.2. Geleceğe Yönerek Geliştirme Önerileri

- **Dinamik Parçalama ve Adaptif Transfer:**

RTT gibi metriklere dayalı olarak blok boyutlarının dinamik bir şekilde ayarlanması, ağ performansına daha iyi yanıt verecek ve veri akışını optimize edecektir.

- **Gelişmiş Hata Düzeltme Protokollerı:**

UDP veri transferinde, FEC gibi ileri tekniklerle hata düzeltme katmanı oluşturulabilir. Böylece yeniden gönderim yükü azaltılarak daha akıcı bir veri akışı sağlanır.

- **Çoklu İstemci Desteği:**

Proje, çoklu istemci bağlantılarını yönetebilecek şekilde genişletilebilir. Böylece, aynı anda birden fazla istemciyle sorunsuz veri alışverişi sağlanabilir ve proje daha ölçeklenebilir hale gelir..

- **Gelişmiş Güvenlik Özellikleri:**

MITM (Man-in-the-Middle) saldırılara karşı gerçek zamanlı IDS entegrasyonu, projenin güvenlik katmanını daha da güçlendirecektir.

- **Hibrit Protokol Seçimi:**

Ağ koşullarına göre TCP/UDP arasında dinamik geçiş özelliği kazandırılarak, en hızlı ve güvenilir iletişim yönteminin anlık olarak seçilmesi sağlanabilir.

## 4.3. Genel Değerlendirme

Bu sınırlamalar, projenin özünü etkilemeyen ama ileriye dönük büyümeye fırsatları sunan alanları temsil etmektedir. Dosya transfer sürecinin güvenilirliği, şifreleme bütünlüğü ve

temel ağ analiz özellikleri projenin güçlü yönleri olarak öne çıkarken; önerilen geliştirme başlıklarları, bu sağlam yapının daha da esnek ve güçlü bir hale getirilmesini hedeflemektedir.

Proje genel hatlarıyla hem güvenli hem de hızlı veri aktarımını sağlama konusunda başarılı olmuş, testlerdeki yüksek performans değerleri ve stabil yapı sayesinde proje hedeflerine ulaşmıştır. Bu sayede, önerilen geliştirmelerle sistemin modern ağ koşullarına daha da uyumlu hale gelmesi mümkün hale gelecektir.

## 5. Sonuç

Bu proje kapsamında, **Gelişmiş Güvenli Dosya Aktarım Sistemi** başarıyla tasarlanmış ve hayata geçirilmiştir. Sistem, hem **ağ performansı** hem de **veri bütünlüğü** konularında yüksek bir güvenilirlik sağlanmış, testler sırasında elde edilen olumlu sonuçlarla proje hedeflerine ulaşmıştır.

Projenin öne çıkan başarılarından bazıları şunlardır:

### **Veri Bütünlüğü ve Güvenliği:**

AES tabanlı simetrik şifreleme, RSA algoritmasıyla anahtar güvenliği ve SHA256 hash doğrulaması sayesinde, dosya transferinde tam veri bütünlüğü ve gizlilik sağlanmıştır. Wireshark analizleri de verinin şifreli akışını ve gizliliğini açıkça ortaya koymuştur.

### **Bağlantı Güvenilirliği:**

TCP protokolü ile yapılan testler, minimum gecikme ve yüksek hızlar elde edilerek bağlantı kalitesinin mükemmel olduğunu göstermiştir. Yerel loopback testlerinde ulaşılan 1.9 Gbit/s değerleri, sistemin temel kapasitesini doğrulamaktadır.

### **UDP ACK/NACK Tabanlı Hata Yönetimi:**

UDP transferlerinde, istemci-sunucu arasındaki ACK/NACK tabanlı hata kontrol mekanizması sayesinde veri kaybı önlenmiş, aktarımın her adımı güvence altına alınmıştır.

### **Performans Analizi ve Grafiksel Raporlama:**

Yapılan RTT ölçümleri ve bant genişliği analizleri (WiFi üzerinden testler dahil), sistemin her aşamada istikrarlı ve güçlü performans gösterdiğini kanıtlamaktadır. Özellikle RTT ve bant genişliği grafiklerinin görsel sunumları, metinsel verilerin somut hale getirilmesini sağlamıştır.

### **Büyük Dosya Transferlerinde Parçalama ve Birleştirme:**

Testlerde 1 MB boyutundaki dosyanın parça bazında transferi başarıyla gerçekleştirilmiş ve “Dosya gönderimi/alımı tamamlandı” çıktıları alınmıştır. Ayrıca SHA256 doğrulama adımları, herhangi bir bozulma veya veri kaybı yaşanmadığını göstermiştir.

### **Esnek ve Geleceğe Uyumlu Altyapı:**

Projede önerilen geliştirme başlıklarları (çoklu istemci desteği, dinamik parçalama, hibrit protokol seçimi gibi) ileriye dönük genişleme ve modernizasyon fırsatlarını açıkça ortaya koymaktadır.

Bu sonuçlar, projenin hem temel hedeflerini hem de ağ ve veri güvenliği gereksinimlerini yüksek bir başarı orANIyla karşıladığı göstermektedir. Gerçek dünyada zayıf ağ bağlantıları, olası paket kayıpları veya performans darboğazları gibi durumlarda bile projenin sağladığı çözüm mimarisi, güçlü bir alternatif olarak öne çıkmaktadır.

Sonuç olarak, projenin **güvenli dosya transferi**, **veri bütünlüğü** ve **ağ performansı** gibi kritik konularda etkili ve sağlam bir çözüm sunduğu net bir şekilde kanıtlanmıştır. Geliştirme önerileriyle desteklenen bu yapı, ileriye dönük uygulamalarda kolayca uyarlanabilir ve farklı kullanım senaryolarına göre ölçeklendirilebilir potansiyele sahiptir.

**Youtube Videosu:** <https://youtu.be/pJFZ6lGiv5M>

## **6. Kaynakça**

### **Python Belgeleri ve Resmi Kütüphaneler:**

- Python 3 Documentation: <https://docs.python.org/3/>
- socket — Low-level networking interface: <https://docs.python.org/3/library/socket.html>
- hashlib — Secure hashes and message digests:  
<https://docs.python.org/3/library/hashlib.html>
- cryptography library: <https://cryptography.io/en/latest/>

### **Ağ Protokollerini ve Performans Testleri:**

- iPerf3 Network Testing Tool: <https://iperf.fr/>
- Wireshark User Guide: [https://www.wireshark.org/docs/wsug\\_html\\_chunked/](https://www.wireshark.org/docs/wsug_html_chunked/)

### **Teorik Kaynaklar ve Makaleler:**

- Tanenbaum, A. S., & Wetherall, D. J. (2010). Computer Networks (5th ed.). Pearson.
- Kurose, J. F., & Ross, K. W. (2017). Computer Networking: A Top-Down Approach (7th ed.). Pearson.

### **Diger Yararlanilan Kaynaklar:**

- Stack Overflow ve çeşitli forumlar (örneğin stackoverflow.com)
- YouTube videoları ve Python socket programlama örnekleri